# CHAPTER 10
# Generative AI and Prompt Engineering

## Introduction

Generative **Artificial Intelligence (AI)** has emerged as one of the most influential and beloved technologies in recent years, particularly since the widespread accessibility of models like ChatGPT to the general public. This powerful technology generates diverse content based on the input it receives, commonly referred as, prompts. As generative AI continues to evolve, it finds applications across various fields, driving innovation and refinement.

Researchers are actively exploring its capabilities, and there **i**s a growing sense that generative AI is inching closer to achieving **Artificial General Intelligence (AGI).** AGI represents the holy grail of AI**,** a system that can understand, learn, and perform tasks across a wide range of domains akin to human intelligence. The pivotal moment in this journey was the introduction of Transformers, a groundbreaking architecture that revolutionized natural language processing. Generative AI, powered by Transformers, has significantly impacted people's lives,

from chatbots and language translation to creative writing and content generation.

In this chapter, we will look into the intricacies of prompt engineering—the art of crafting effective inputs to coax desired outputs from generative models. We will explore techniques, best practices, and real-world examples, equipping readers with a deeper understanding of this fascinating field.

## Structure

In this chapter, we will discuss the following topics:

- Generative AI
- Large language model
- Prompt engineering and types of prompts
- Open-ended prompts vs. specific prompts
- Zero-shot, one-shot, and few-shot learning
- Using LLM and generative AI models
- Best practices for building effective prompts
- Industry-specific use cases

## Objectives

By the end of this chapter, you would have learned the concept of generative AI, prompt engineering techniques, ways to access generative AI, and many examples of writing prompts.

## Generative AI

Generative AI is an artificially intelligent computer program that has a remarkable ability to create new content, and the content is sometimes fresh and original artifacts. It can generate audio, images, text, video, code, and more. It produces new things based on what it has

learned from existing examples.

Now, let us look at how generative AI is built. They leverage powerful foundation models trained on massive datasets and then fine-tuned with complex algorithms for specific creative tasks. Generative AI is based on four major components: the foundation model, training data, fine-tuning, complex mathematics, and computation. Let us look at them in detail as follows:

- Foundation models are the building blocks. Generative AI often relies on foundation models, such as **Large Language Models (LLMs).** These models are trained on large amounts of text data, learning patterns, context, and grammar.
- Training data is a large reference database of existing examples. Generative AIs learn from training data, which includes everything from books and articles to social media posts, reports, news articles, dissertations, etc. The more diverse the data, the better they become at generating content.
- After initial training, the models undergo fine-tuning. Fine-tuning customizes them for specific tasks. For example, GPT-4 can be fine-tuned to generate conversational responses or to write poetry.
- Building these models involves complex mathematics and requires massive computing power. However, at their core, they are essentially predictive algorithms.

## Understanding generative AI

This generative AI takes in the prompt. You provide a prompt (a question, phrase, or topic). Based on the input prompt, AI uses its learned patterns from training data to generate an answer. It does not just regurgitate existing content; it creates something new. The two main approaches used by generative AI are **Generative**

**Adversarial Networks (GANs)** and autoregressive models:

- **GANs:** Imagine two AI models competing against each other. One, the generator, tries to generate realistic data (images, text, etc.), while the other, the discriminator, tries to distinguish the generated data from real data. Through this continuous competition, the generator learns to produce increasingly realistic output.
- **Autoregressive models:** These models analyze sequences of data, such as sentences or image pixels. They predict the next element in the sequence based on the previous ones. This builds a probabilistic understanding of how the data is structured, allowing the model to generate entirely new sequences that adhere to the learned patterns.

Beyond the foundational models such as GANs and autoregressive models, generative AI also relies on several key mechanisms that enable it to process and generate sophisticated outputs. Behind the scenes, generative AI performs embedding and uses attention mechanism. These two critical components are described as follows:

- **Embedding:** Complex data such as text or images are converted into numerical representations. Each word or pixel is assigned a vector containing its characteristics and relationships to other elements. This allows the model to efficiently process and manipulate the data.
- **Attention mechanisms:** In text-based models, attention allows the AI to focus on specific parts of the input sequence when generating output. Imagine reading a sentence; you pay more attention to relevant words for comprehension. Similarly, the model prioritizes critical elements within the input prompt to create a coherent response.

While understanding generative AI is crucial, it is equally important to keep the human in the loop. Human validation and control are essential to ensure the reliability and ethical use of AI systems. Even though generative AI can produce impressive results, it is not perfect. Human involvement remains essential for validation and control. Validation is when AI-generated content requires human evaluation to ensure accuracy, factuality, and lack of bias. Control is when humans define the training data and prompts that guide the AI's direction and output style.

## Large language model

**Large Language Model (LLM)** is a kind of AI program that excels in understanding and generating human language. It carries out certain functionalities based on trained data, and it consists of multiple building blocks which consists of technologies like deep learning, transformers, and many more.

Following is a brief description of the three aspects:

- **Function:** LLMs can recognize, summarize, translate, predict, and generate text content. They are like super-powered language processors.
- **Training:** They are trained on massive amounts of text data, which is why they are called LLMs. This data can come from books, articles, code, and even conversations.
- **Building blocks:** LLMs are built on a special type of machine learning called deep learning, and more specifically on a neural network architecture called a transformer model.

Now, let us look at how LLMs work. It is the same as generative AI, they take input, encode it and decode it to answer the input. The LLM receives text input, like a sentence or a question. Encoding is a transformer model

within the LLM that analyzes the input, recognizing patterns and relationships between words. Finally, decoding is based on the encoded information; the LLM predicts the most likely output, which could be a translation, a continuation of the sentence, or an answer to a question.

## Prompt engineering and types of prompts

Prompt engineering is writing, refining, and optimizing prompts to achieve flawless human-AI interaction. It also entails keeping an updated prompt library and continuously monitoring those prompts. It is like being a teacher for the AI, guiding its learning process to ensure it provides the most accurate and helpful responses. For example: Imagine you are teaching a child to identify animals. You show them a picture of a dog and say, *this is a dog*. The child learns from this and starts recognizing dogs. This is similar to how AI learns from prompts. Now, suppose the child sees a wolf and calls it a dog. This is where refinement comes in. You correct the child by saying, no, that is not a dog; it is a wolf.

Similarly, in prompt engineering, we refine and optimize the prompts based on the AI's responses to make the interaction more accurate. Monitoring is like keeping an eye on the child is learning progress. If the child starts calling all four-legged animals' dogs, you know there is a problem. Similarly, prompt engineers continuously monitor the AI's responses to ensure it is learning correctly. Maintaining an up-to-date prompt library is like updating the child's knowledge as they grow. As the child gets older, you might start teaching them about different breeds of dogs.

Similarly, prompt engineers update the AI's prompts as it learns and grows, ensuring it can handle more complex tasks and inquiries. Prompt design is both an art and a

science. Experimenting, iterating, and refining your approach to unlock the full potential of AI-generated responses across applications is a secret to prompting. Whether you are a seasoned developer or a curious beginner, understanding prompt types is crucial for generating insightful and relevant responses. Now, let us understand the types of prompts.

## Open-ended prompts versus specific prompts

Open-ended prompts are broad and flexible, giving the AI system room to generate diverse content. They allow for creativity and exploration**,** encourage imaginative responses without strict constraints. For example, *write a short story about a mysterious mountain* is an open-ended prompt where the AI system is free to use creativity as there are no constraints.

On the other hand, **specific prompts** provide clear instructions and focus on a specific task or topic. They guide the AI to a specific result. They are useful when you need precise answers or targeted information. For example, *summarize the key findings of the research paper titled Climate Change Impact on Arctic Ecosystems*. The choice of open-ended and specific prompts depends on the desired outcome and objective of the task. However, clear and specific prompts provide more accurate and relevant content. *Table 10.1* provides the domain in which each of the above prompt types will be useful, along with the examples of each:

| Types | Useful domain | Examples |
|---|---|---|
| Open-Ended | Creative writing | • Write a short story about an unlikely friendship between a human and an AI in a futuristic city.<br>• Imagine a world where gravity works differently. Describe the daily life of someone living in this world. |
| | Brainstorming | • Generate ideas for a new sci-fi movie plot involving time travel.<br>• List five innovative uses for drones beyond photography and surveillance. |
| | Exploration and imagination | • Describe an alien species with unique physical features and cultural practices.<br>• Write a poem inspired by the colors of a sunset over a tranquil lake. |
| | Character development | • Create a detailed backstory for a rogue archaeologist who hunts ancient artifacts.<br>• Introduce a quirky sidekick character who communicates only through riddles. |
| | Philosophical reflection | • Explore the concept of free will versus determinism in a thought-provoking essay.<br>• Discuss the ethical implications of AI achieving consciousness. |

| Types | Useful domain | Examples |
|---|---|---|
| Specific Prompts | Summarization | • Provide a concise summary of the American Civil War in three sentences.<br>• Summarize the key findings from the World health statistics 2023 report. |
| | Technical writing | • Write step-by-step instructions for setting up a home network router.<br>• Create a user manual for a smartphone camera app, including screenshots. |
| | Comparisons and contrasts | • Compare and contrast the advantages of electric cars versus traditional gasoline cars.<br>• Analyze the differences between classical music and contemporary pop music. |
| | Problem-solving | • Outline a Python code snippet to calculate the Fibonacci sequence.<br>• Suggest strategies for reducing plastic waste in a coastal city. |
| | Persuasive writing | • Compose an argumentative essay advocating for stricter regulations on social media privacy.<br>• Write a letter to the editor supporting the implementation of renewable energy policies. |

***Table 10.1:*** *Prompts types, with their use and examples*

## Zero-shot, one-shot, and few-shot learning

Prompting techniques play a key role in shaping the behavior of LLMs. They allow prompts to be designed and optimized for better results. These techniques are essential for eliciting specific responses from generative AI models or LLMs. Zero-shot, one-shot, and few-shot prompting are common prompting techniques. Besides that, the chain of thought, self-consistency, generated knowledge prompting, and retrieval augmented generation are additional strategies.

## Zero-shot

Used when no labeled data is available for a specific task. It is useful, as it enables models to generalize beyond their training data by learning from related information. For example, recognizing new classes without prior examples, for example, identifying exotic animals based on textual descriptions). Now, let us look at a few more examples as follows:

**Example 1**:

- **Prompt**: **Translate the following English sentence to French: The sun is shining.**
- **Technique**: Zero-shot prompting allows the model to perform a task without specific training. The model can translate English to French even though the exact sentence was not seen during training.

**Example 2**:

- **Prompt**: **Summarize the key points from the article about climate change.**
- **Technique**: Zero-shot summarization. The model generates a summary without being explicitly trained on the specific article.

## One-shot

It is used to deal with limited labeled data and is ideal for scenarios where many labeled examples are scarce. For example, training models with only one example per class, for example, recognition of rare species or ancient scripts. In one-shot learning, a model is expected to understand and generate a response or task (such as writing poem) based on a single prompt without needing additional examples or instructions. Now, let us look at a few examples as follows:

**Example 1**:

- **Prompt**: **Write a short poem about the moon.**
- **Technique**: A single input prompt is given to generate content.

**Example 2**:

- **Prompt: Describe a serene lakeside scene.**
- **Technique**: Model is given one-shot description (i.e, a vivid scene) in the given prompt.

## Few-shot

The purpose of few-shot learning is that it can learn from very few labeled samples. Hence, it is useful to bridge the gap between one-shot and traditional supervised learning. For example, it addresses tasks such as medical diagnosis with minimal patient data or personalized recommendations. Now, let us look at a few examples:

**Example 1**:

- **Prompt: Continue the story: Once upon a time, in a forgotten forest**
- **Technique:** Few-shot prompting allows the model to build on a partial narrative.

**Example 2**:

- **Prompt: List three benefits of meditation.**
- **Technique :** Few-shot information retrieval. The model provides relevant points based on limited context.

## Chain-of-thought

**Chain-of-Thought (CoT)** encourages models to maintain coherent thought processes across multiple responses. It is useful for generating longer, contextually connected outputs. For example, crafting multi-turn dialogues or essay-like responses. Now, let us look at a few examples as follows:

**Example 1**:

- **Prompt: Write a paragraph about the changing seasons.**
- **Technique:** Chain of thought involves generating coherent content by building upon previous sentences. Here, writing about the change in the season involves keeping the past season in mind.

**Example 2**:

- **Prompt: Discuss the impact of technology on human relationships.**
- **Technique:** Chain of thought essay. The model elaborates on the topic step by step.

## Self-consistency

Self-consistency prompting is a technique used to ensure that a model's responses are coherent and consistent with its previous answers. This method plays a crucial role in preventing the generation of contradictory or nonsensical information, especially in tasks that require logical reasoning or factual accuracy. The goal is to make sure that the model's output follows a clear line of thought and maintains internal harmony. For instance, when performing fact-checking or engaging in complex reasoning, it's vital that the model doesn't contradict itself within a single response or across multiple responses. By applying self-consistency prompting, the model is guided to maintain logical coherence, ensuring that all parts of the response are in agreement and that the conclusions drawn are based on accurate and consistent information. This is particularly important in scenarios where accuracy and reliability are key, such as in medical diagnostics, legal assessments, or research. Now, let us look at a few examples s follows:

**Example 1**:

- **Prompt: Create a fictional character named Gita and describe her personality.**

- **Technique:** Self-consistency will ensure coherence within the generated content.

**Example 2**:

- **Prompt: Write a dialogue between two friends discussing their dreams.**
- **Technique:** Self-consistent conversation. The model has to maintain character consistency throughout.

## Generated knowledge

Generated knowledge prompting encourages models to generate novel information. It is useful for creative writing, brainstorming, or expanding existing knowledge. For example, crafting imaginative stories, inventing fictional worlds, or suggesting innovative ideas. Since this is one of the areas of keen interest for most researchers, efforts are being put to make it better for generating knowledge. Now, let us look at a few examples as follows:

**Example 1**:

- **Prompt: Explain the concept of quantum entanglement.**
- **Technique:** Generated knowledge provides accurate information.

**Example 2**:

- **Prompt: Describe the process of photosynthesis.**
- **Technique:** Generated accurate scientific explanation.

## Retrieval augmented generation

**Retrieval augmented generation (RAG)** combines generative capabilities with retrieval-based approaches. It enhances content by pulling relevant information from external sources. For example, improving the quality of responses by incorporating factual details from existing knowledge bases. Now, let us look at a few examples as

follows:

**Example 1**: Generating friendly ML paper titles

- **Prompt: Create a concise title for a machine learning paper discussing transfer learning.**
- **Technique:** RAG combines an information retrieval component with a text generator model.
- **Process:**
    - RAG retrieves relevant documents related to transfer learning (e.g., research papers, blog posts).
    - These documents are concatenated as context with the original input prompt.
    - The text generator produces the final output.

**Example 2**: Answering complex questions with external knowledge

- **Prompt: Explain the concept of quantum entanglement.**
- **Technique:** RAG leverages external sources for example, Wikipedia to ensure factual consistency.
- **Process:**
    - RAG retrieves relevant Wikipedia articles on quantum entanglement.
    - The retrieved content is combined with the original prompt.
    - The model generates an accurate explanation.

## Using LLM and generative AI models

Using generative AI and LLM has become very simple and easy. Here, we give a quick overview of using them with Python in Jupyter Notebook. Also, by pointing to some of the existing web application and their **Uniform Resource**

**Locator (URLs).** Now, there are many LLM, but the GPT-4 seems dominant; besides that, *Google's Gemini*, *Meta's Llama 3*, *X's Grok-1.5*, open-source models in hugging face exist, and growth continues. Now let us have a look at the following:

## Setting up GPT-4 in Python using the OpenAI API

Follow these steps to setup GPT-4 in Python using the OpenAI API:

1. Create an OpenAI developer account
    a. Before understanding the technical details, you need to create an account with OpenAI. Follow these steps:
        i. Go to the API signup page.
        ii. Sign up with your email address and phone number.
        iii. Once registered, go to the API keys page.
        iv. Create a new secret key (make sure to keep it secure).
        v. Add your debit or credit card details on the **Payment Methods** page.
2. Install required libraries
    a. To use GPT-4 via the API, you will need to install the OpenAI library. Open your command prompt or terminal and run the following command:
        1. `pip install openai`
3. Securely store your API keys
    a. Keep your secret API key confidential. One of the easy to avoid hardcoding Open-AI API key directly in the code, is by using **dotenv**.
    b. Install the **python-dotenv** package:
        1. `pip install python-dotenv`
    c. Create a **.env** file in the project directory and add an

API key on it:

```
OPENAI_API_KEY=your_actual_api_key_here
```

d. In your Python script or Jupyter Notebook, load the API key from the **.env** file.

4. Start generating content with GPT-4.

**Tutorial 10.1:** To use and access GPT-4 using Open-AI API key, install **openai** and **dotenv** (which is to hide API key from Jupyter Notebook) and then follow the code:

```
# Import libraries
import os
from dotenv import load_dotenv
import openai
from IPython.display import display, Markdown
# Load environment variables from .env
load_dotenv()

# Get the API key
openai.api_key = os.getenv("OPENAI_API_KEY")
# Create completion using GPT-4
completion = openai.ChatCompletion.create(
    model="gpt-4",
    messages=[
        {"role": "user", "content": "What is artificial intelligence?"}
    ]
)
# Print the response
print(completion.choices[0].message['content'])
```

Most of you might have used the GPT-3, which can be accessed free of cost. GPT-4 can be accessed by paying from **https://chat.openai.com/**. It can also be used with

Microsoft Copilot or notebook (**https://copilot.microsoft.com/**).

Similarly, *Gemini* from *Google* can be accessed using **(https://gemini.google.com/app)**. Also, from the hugging face platform, many open-source models can be accessed and used.

**Tutorial 10.2**: To use open-source model **google/flan-t5-large'** from Hugging face for text generation, first install **langchain**, **huggingface_hub**, **transformers**, and then type the following code:

```
from langchain import PromptTemplate, HuggingFaceHub, LLMChain
import os
os.environ['HUGGINGFACEHUB_API_TOKEN'] = 'REPLACE_WITH_HUGGINGFACE_TOKEN_KEY'
prompt = PromptTemplate(input_variables=[
                    'Domain'], template='What are the uses of artificial intelligence in {Domain} ?')
chain = LLMChain(llm=HuggingFaceHub(
   repo_id='google/flan-t5-large'), prompt=prompt)
print(chain.run(Domain='healthcare'))
```

Running above code the '**google/flan-t5-large'** model will give a reply to the question. In this way any model can be accessed and used from HuggingFace platform.

## Best practices for building effective prompts

To write an effective prompt, the task for which the prompt has to be written, the context, example, persona, format, and tone of the prompt are important. Of these, the task is a must, and giving context to it is kind of mandatory. Having examples is important. Besides these, having persona, format, and tone are good to have in a prompt.

Also, do not be afraid to ask for creative results to solve technically savvy problems to LLMs. They are creative and can produce poems, stories, and even jokes. They can also process mathematical problems; you can ask them to compute mathematical, logical problems.

For example, in a sentence: **I am a tenth grader student, define what artificial intelligence is to me?** The first part of the sentence, **I am a tenth grader student** is context and defines artificial intelligence is the task. Basically, to build effective prompts for LLMs or generative AI, it is good to know and keep the following in mind:

- **Task:** Task is compulsory to include in the prompt, use an action verb to specify the task. Write, Create, Give, Analyze, Calculate, etc. are the action verbs.
    - For example, in a prompt, **Write a short essay on the impact of climate change.**
        - Here, **write** is the action verb, and the task is to write an essay. The prompt can have any number of tasks like it can have a single task or multiple tasks in one.
    - For example, **Calculate the area of a circle with a radius of 5 units is a single task prompt.**
        - Similarly, **Calculate the area of a circle with a radius of 5 units and the circumference of the same circle** is a multitask prompt.
- **Context:** Providing context includes providing background information, setting the environment, and mentioning what the desired success or outcome looks like. For example, **As a high school student, write a letter to your local government official about improving public transportation in the city**. Here, the context is that the user is a high school student

writing to a government official.

- **Clarity and specificity:** Ensure the prompt is clear and unambiguous. Being specific in your prompt will guide the LLM toward the desired output. For example, instead of saying **Write about a historical event, you could say Write a detailed account of the Battle of Kurukshetra.**
- **Iterative refinement:** It is often beneficial to refine the prompts iteratively based on the model's responses. For example, if the model's responses are too broad, the prompt can be made more specific.
- **Exemplars, persona, format and tone matters:**
    - Exemplars provide examples to teach the LLM to answer correctly through example or suggestions. For example, **Write a poem about spring**. For instance, you could talk about blooming flowers, longer days, or the feeling of warmth in the air.
    - Persona prompts make the LLM think of someone you wish to have for the task you are facing. For example, **Imagine you are a tour guide. Describe the main attractions of New Delhi.**
    - Format is how you want your output to be structured, and the tone is the mood or attitude conveyed in the response. For example, **Write a formal email to your professor asking for an appointment. The tone should be respectful and professional.**

## Industry-specific use cases

Generative AI and LLMs have found applications in many fields and domains. However, their true power lies in fine-tuning- tailoring these models for specific use cases or industries. By customizing prompts and constraints,

organizations can harness the power of LLMs to address unique challenges in different fields as described here:

- **Engineering:** In engineering, code generation, optimizing design, and solving logical problems are a few examples of using generative AI. For example, software engineers can use prompts to automate repetitive coding tasks, freeing time to solve complex problems, etc. Such as, writing a Python function to calculate the area of a circle. Mathematicians and programmers can use it to solve complex mathematical problems like solving and implementing a quadratic equation of the form *ax^2 + bx + c = 0*. It can also be used to create variations of a design based on specified parameters. An engineer could provide a prompt outlining desired material properties, weight constraints, and functionality for a bridge component. The AI would then generate multiple design options that meet these criteria, accelerating the design exploration process.
- **Healthcare**:
    - **Personalized patient education:** Imagine an AI that creates educational materials tailored to a patient's specific condition and literacy level. A physician could ask the AI to create a video explaining diabetes management in a language appropriate for an elderly patient. This can improve patient understanding and medication adherence.
    - **Drug discovery:** Developing new drugs is a long and expensive process. Generative AI can be directed to design new drug molecules with specific properties to target a particular disease. This can accelerate drug discovery and potentially lead to breakthroughs in treatment.
    - **Mental health virtual assistants:** AI-powered

chatbots can provide basic mental health support and emotional monitoring. Prompts can guide the AI to provide appropriate responses based on a user's input, providing 24/7 support and potentially reducing the burden on human therapists.

- **Education:**
  - **Personalized learning materials:** Generative AI can create customized practice problems, quizzes, or educational games based on a student's strengths and weaknesses. A teacher could ask the AI to generate math practice problems of increasing difficulty for a student struggling with basic algebra.
  - **Simulate real-world scenarios:** Training future doctors, nurses, or even firefighters require exposure to a variety of situations. Generative AI can be instructed to create realistic simulations of medical emergencies, allowing students to practice decision-making in a safe environment.
  - **Create accessible learning materials:** Generative AI can be asked to create alternative formats for educational content, such as converting text into audio descriptions for visually impaired students or generating sign language translations for lectures.
- **Manufacturing:**
  - **Create bills of materials (BOMs):** Creating BOMs which list all the components needed for a product, can be a tedious task. Generative AI, prompted by a product design or 3D model, can automatically generate a detailed BOM, improving efficiency and reducing errors.
  - **Predictive maintenance:** By analyzing sensor

data and historical maintenance records, generative AI models can be asked to predict when equipment might fail. This enables proactive maintenance, minimizing downtime and lost production.

- **Content creation:**
    - **Generate product descriptions:** E-commerce businesses can use generative AI to create unique and informative product descriptions based on product specifications and customer data. A prompt could include details such as product features, target audience, and desired tone of voice.
    - **Write marketing copy:** Creating catchy headlines, social media posts, or email marketing content can be time-consuming. Generative AI can be prompted with a product or service and generate multiple creative copy options, allowing marketers to choose the most effective.

## Conclusion

The field of generative AI, driven by LLMs, is at the forefront of technological innovation. Its impact is reverberating across multiple domains, simplifying tasks, and enhancing human productivity. From chatbots that engage in natural conversations to content generation that sparks creativity, generative AI has become an indispensable ally. However, this journey is not without its challenges. The occasional hallucination where models produce nonsensical results, the need for alignment with human values, and ethical considerations all demand our attention. These hurdles are stepping stones to progress. Imagine a future where generative AI seamlessly assists us, a friendly collaborator that creates personalized emails, generates creative writing, and solves complex problems. It

is more than a tool; it is a companion on our digital journey.

This chapter serves as a starting point- an invitation to explore further. Go deeper, experiment, and shape the future. Curiosity will be your guide as you navigate this ever-evolving landscape. Generative AI awaits your ingenuity, and together, we will create harmonious technology that serves humanity.

In final *Chapter 11*, *Data Science in Action: Real-World Statistical Applications*, we explore two key projects. The first applies data science to banking data, revealing insights that inform financial decisions. The second focuses on health data, using statistical analysis to enhance patient care and outcomes. These real-world applications will demonstrate how data science is transforming industries and improving lives.

## Join our book’s Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

**https://discord.bpbonline.com**

# CHAPTER 11
# Real World Statistical Applications

## Introduction

As we reach the climax of the book, this final chapter serves as a practical bridge between theoretical knowledge and real-world applications. Throughout this book, we have moved from the basics of statistical concepts to advanced techniques. In this chapter, we want to solidify your understanding by applying the principles you have learned to real-world projects. In this chapter, we will delve into two comprehensive case studies-one focused on banking data and the other on healthcare data. These projects are designed not only to reinforce the concepts covered in earlier chapters but also to challenge you to use your analytical skills to solve complex problems and generate actionable insights. By implementing the statistical methods and data science techniques discussed in this book, you will see how data visualization, exploratory analysis, inferential statistics and machine learning come together to solve real-world problems. This hands-on approach will help you appreciate the power of statistics in data science and prepare you to apply these skills in your future endeavors,

whether in academia or industry. The final chapter puts theory into practice, ensuring that you leave with both the knowledge and the confidence to tackle statistical data science projects on your own.

## Structure

In this chapter, we will discuss the following topics:

- Project I: Implementing data science and statistical analysis on banking data
- Project II: Implementing data science and statistical analysis on health data

## Objectives

This chapter aims to demonstrate the practical implementation of data science and statistical concepts using real-world synthetic banking and health data generated for this book only, as a case study. By analyzing these datasets, we will illustrate how to derive meaningful insights and make informed decisions based on statistical inference.

## Project I: Implementing data science and statistical analysis on banking data

Project I harnesses the power of synthetic banking data to explore and analyze customer behaviors and credit risk profiles in the banking sector. The generated synthetic dataset contains detailed information on customer demographics, account types, transaction details, loan and credit cards, as shown in *Figure 11.1*:

| | CustomerID | Age | Gender | Occupation | EducationLevel | MaritalStatus |
|---|---|---|---|---|---|---|
| 0 | abc7b464-6e65-4303-9e51-eeaf4f3d1ca2 | 46 | Other | Production designer, theatre/television/film | Master | Single |
| 1 | 41f7f53a-b53f-4213-906c-98e24ec65c6c | 37 | Male | Radio producer | High School | Widowed |

| | CustomerID | AccountNumber | AccountType | DateAccountOpened | AccountStatus |
|---|---|---|---|---|---|
| 0 | abc7b464-6e65-4303-9e51-eeaf4f3d1ca2 | 270749932 | Savings | 2009-06-08 | Active |
| 1 | 41f7f53a-b53f-4213-906c-98e24ec65c6c | 6989827612 | Savings | 2012-11-25 | Active |

| | TransactionID | AccountNumber | DateofTransaction | TypeofTransaction | Amount | TransactionDescription |
|---|---|---|---|---|---|---|
| 0 | 73018798 | 270749932 | 2022-08-13 | Deposit | 2984.87 | And drop less seat strategy eye. |
| 1 | 41549331 | 6989827612 | 2022-12-25 | Transfer | 5601.25 | Strategy instead study job list. |

| | LoanID | AccountNumber | LoanAmount | LoanStartDate | LoanType | InterestRate | LoanTerm | MonthlyRepaymentAmount |
|---|---|---|---|---|---|---|---|---|
| 0 | 6342289 | 270749932 | 14984 | 2010-04-10 | Auto | 7.15 | 30 | 1669 |
| 1 | 46972378 | 6989827612 | 23867 | 2021-02-05 | Personal | 3.44 | 30 | 1749 |

| | CardNumber | AccountNumber | CreditLimit | CardType | ExpiryDate | CVV |
|---|---|---|---|---|---|---|
| 0 | 213131988105207 | 270749932 | 3083 | MasterCard | 2026-05-09 | 036 |
| 1 | 4810457564935559469 | 6989827612 | 8873 | MasterCard | 2025-12-09 | 014 |

***Figure 11.1****: Structure of the synthetic health data*

Then, using the above-mentioned synthetic health data, follow these steps:

1. **Data loading and exploratory analysis**: The initial phase involves loading the synthetic banking data followed by exploratory analysis. We employ descriptive statistics and visualization techniques to understand the underlying patterns and distributions within the data, here we show distribution of customer by ages, and account types.
2. **Statistical testing**: The next step focuses on statistical analysis to explore the relationships between variables. Here we analyze relationship between customer's education level and their chosen account types. This involves assessing whether significant differences exist in the type of accounts held, based on education levels.
3. **Credit card risk analysis:** By leveraging customer data attributes like education level, marital status, account type, loan type, interest rate, and credit limit, we categorize customers into risk groups: high, medium, and low. This segmentation is based on predefined criteria that consider both financial behavior and demographic factors.

4. **Predictive modelling**: The core analytical task involves developing a predictive model to classify customers into risk categories (high, medium, low) for issuing credit cards. This model helps in understanding and predicting customer risk profiles based on their banking and demographics information, to decide if it is right decision to issue the credit card.
5. **Model deployment for user input prediction:** In the final step, the trained model is deployed as a tool that accepts user inputs via the command line for attributes such as education level, marital status, account type, loan type, interest rate, and credit limit. This allows for real-time risk assessment and prediction on potential credit card issuance.

This project not only enhances our understanding of customer behaviors and risk but also aids in strategic decision-making for credit issuance based on robust data-driven insights.

## Part 1: Exploratory data analysis

Here, we will examine the data to understand distributions and relationships. The code snippet for the **Exploratory Data Analysis (EDA)** as follows:

```
# Load data from CSV files
customers = pd.read_csv(
    '/workspaces/ImplementingStatisticsWithPython/notebooks/chapter11project/banking/data/customers.csv')
accounts = pd.read_csv(
    '/workspaces/ImplementingStatisticsWithPython/notebooks/chapter11project/banking/data/accounts.csv')
# Plotting the age distribution of customers
plt.figure(figsize=(10, 6))
plt.hist(customers['Age'], bins=30, density=True, alpha=0.5, color='blue')
```

```
plt.title('Distribution of Customer Ages')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.savefig('age_distribution.jpg', dpi=300, bbox_inches='tight')
plt.show()
# Analyzing account types by plotting distribution in bar plot
account_types = accounts['AccountType'].value_counts()
plt.figure(figsize=(10, 6))
plt.bar(account_types.index, account_types.values, alpha=0.5, color='blue')
plt.title('Distribution of Account Types')
plt.xlabel('Account Type')
plt.ylabel('Number of Accounts')
plt.savefig('account_type_distribution.jpg', dpi=300, bbox_inches='tight')
plt.show()
```

Following is the output of *Project I*, part 1:

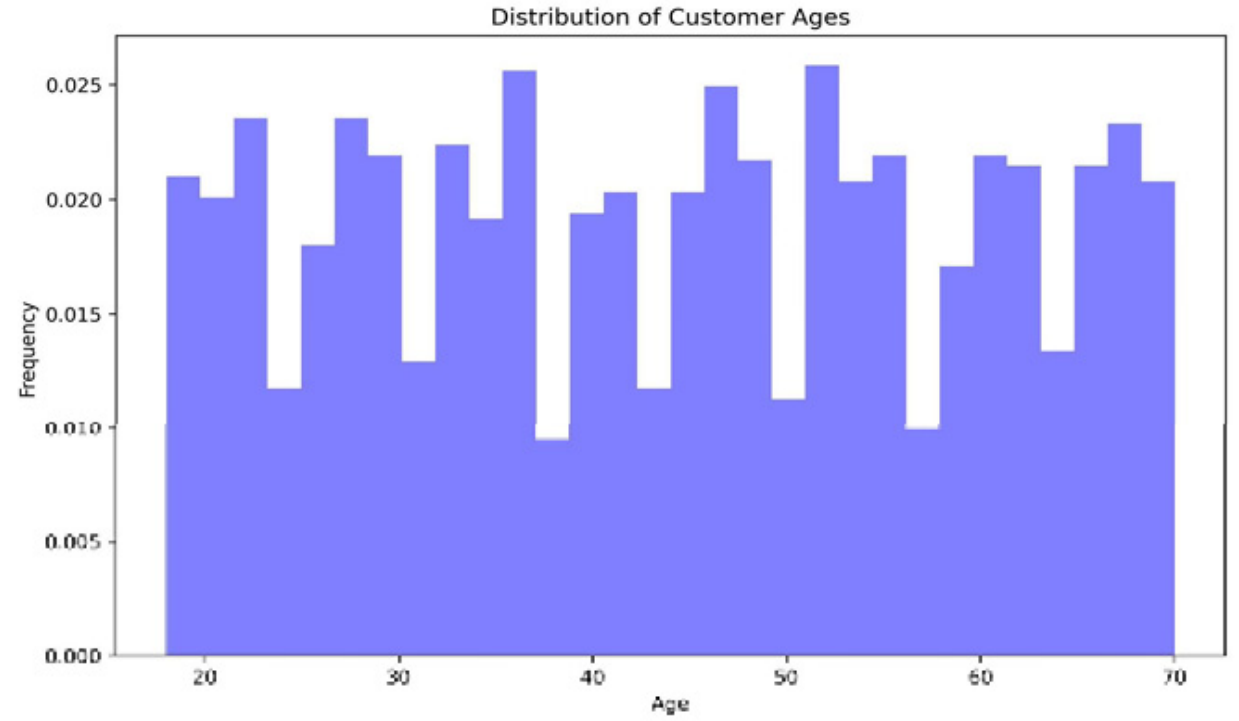


***Figure 11.2**: Distribution of customer ages*

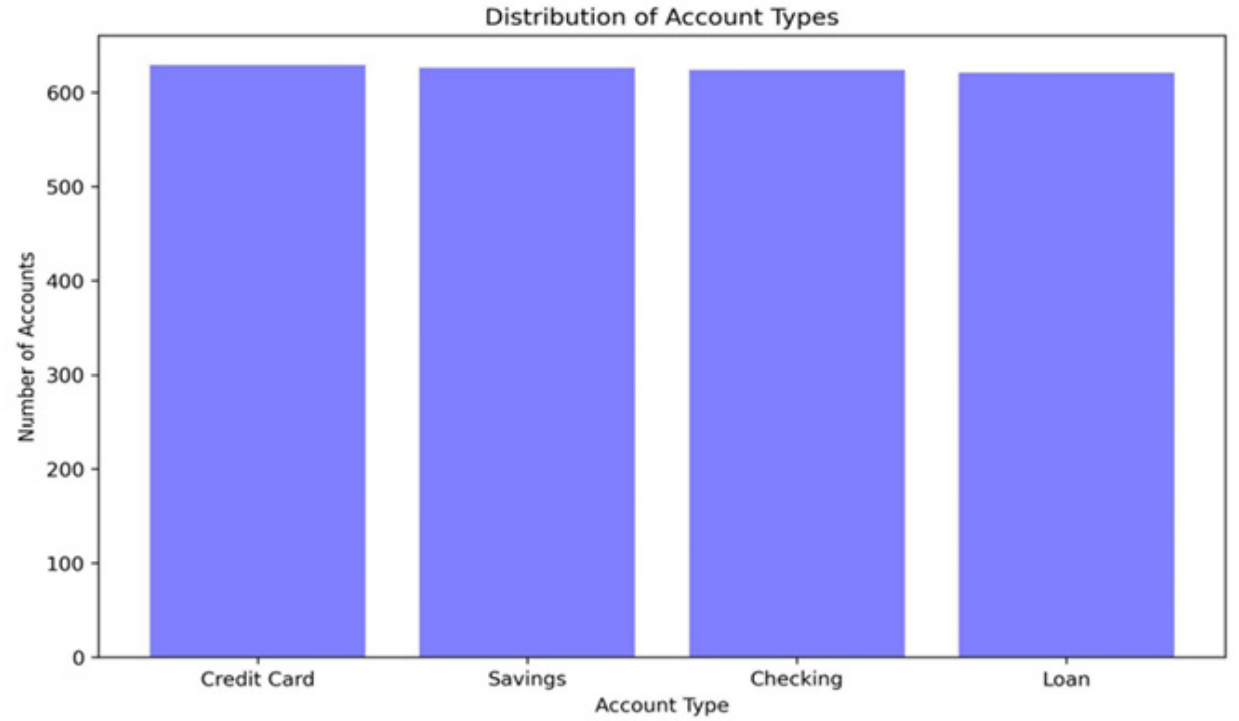


***Figure 11.3:** Distribution of customers by age in histogram and account type in bar chart*

*Figure 11.3* shows there is a consistent distribution of customers by account type and by age, ranging from approximately 18 to 70 years old.

## Part 2: Statistical testing

We will perform statistical tests to examine differences in account types based on education level. For this, we perform chi square test, compute the p-value and the expected frequencies. Expected frequencies represent the hypothetical counts of observations within each category or combination if there were no association between the variables being studied.

For example, Let us say you are studying the relationship between favorite ice cream flavor (chocolate, vanilla, strawberry) and gender (male, female). If there is no connection between favorite flavor and gender, you would expect an equal distribution of flavors among both males and females. So, the expected frequencies would be roughly equal for each combination of flavor and gender but if the expected frequency distribution is not equal. Then you might find that chocolate is much more popular among males than females, and strawberry is more popular among females than males, suggesting that there might be a relationship between ice cream flavor preference and gender. The code snippet for the statistical testing is as follows:

```
# Creating a contingency table of account type by education level
contingency_table = pd.crosstab(
    accounts['AccountType'], customers['EducationLevel'])
# Chi-squared test
chi2, p, dof, expected = chi2_contingency(contingency_table)
# Printing results with labels for better readability
print("Chi-squared Test results:")
```

```
print(f"P-value: {p}")
print("\nExpected Frequencies:")
# Printing expected frequencies with proper labels for easy understanding
expected_df = pd.DataFrame(expected,
                           index=contingency_table.index,
                           columns=contingency_table.columns)
print(expected_df)
```

Following is the output of *Project I*, part 2:

```
Chi-squared Test results:
P-value: 0.4387673577903144
Expected Frequencies:
EducationLevel  Bachelor  High School    Master       PhD
AccountType
Checking         154.9152    156.2192  173.1712  167.6944
Credit Card      143.0352    144.2392  159.8912  154.8344
Loan             143.2728    144.4788  160.1568  155.0916
Savings          152.7768    154.0628  170.7808  165.3796
```

Output shows p-value above the standard significance level of 0.05, so we cannot reject the null hypothesis, indicating no significant association between **Account Type** and **Education Level**. Larger expected frequencies signify higher anticipated counts, suggesting a greater likelihood of those outcomes under the assumption of independence between variables, while smaller values imply lower expected counts.

## Part 3: Analyze the credit card risk

Now, we will analyze the credit card risk based on education level, marital status, account type, loan type,

interest rate, credit limit. To do so, we will create a comprehensive dataset that includes various attributes from different **Comma Separated Value (CSV)** files and categorizing credit card risk into high, medium, and low based on several factors.

We use the following conditions and features to analyze what is the risk for issuing a credit card:

- **Interest rate**: High risk if the interest rate are high.
- **Credit limit**: High risk if the credit limit is high.
- **Education level**: Higher educational levels might correlate with lower risk.
- **Marital status**: Married individuals might be considered lower risk compared to single ones.
- **Account type**: Certain account types like loans might carry higher risk than others like savings.
- **Loan amount**: Larger loan amounts could be considered higher risk.

The following code creates a comprehensive dataset by merging useful datasets and applying conditions to categorize risk, and saving the new dataset with a column credit card risk type:

```
# Merge customers with accounts
customer_accounts = pd.merge(customers, accounts, on='CustomerID', how='inner')
# Merge the above result with loans
customer_accounts_loans = pd.merge(
    customer_accounts, loans, on='AccountNumber', how='inner')
# Merge the complete data with credit cards
complete_data = pd.merge(customer_accounts_loans,
                         credit_cards, on='AccountNumber', how='inner')
# Function to categorize credit card risk, using the cond
```

```
itions
def categorize_risk(row):
    # Base risk score initialization
    risk_score = 0
    # Credit Limit and Interest Rate Conditions
    if row['CreditLimit'] > 7000 or row['InterestRate'] > 7
:
        risk_score += 3
    elif 5000 < row['CreditLimit'] <= 7000 or 5 < row['Int
erestRate'] <= 7:
        risk_score += 2
    else:
        risk_score += 1
    # Education Level Condition
    if row['EducationLevel'] in ['PhD', 'Master']:
        risk_score -= 1  # Lower risk if higher education
    elif row['EducationLevel'] in ['High School']:
        risk_score += 1  # Higher risk if lower education
    # Marital Status Condition
    if row['MaritalStatus'] == 'Married':
        risk_score -= 1
    elif row['MaritalStatus'] in ['Single', 'Divorced', 'Wido
wed']:
        risk_score += 1
    # Account Type Condition
    if row['AccountType'] in ['Loan', 'Credit Card']:
        risk_score += 2
    elif row['AccountType'] in ['Savings', 'Checking']:
        risk_score -= 1
    # Loan Amount Condition
    if row['LoanAmount'] > 20000:
```

```
        risk_score += 2
    elif row['LoanAmount'] <= 5000:
        risk_score -= 1
    # Categorize risk based on final risk score
    if risk_score >= 5:
        return 'High'
    elif 3 <= risk_score < 5:
        return 'Medium'
    else:
        return 'Low'
# Apply the function to determine credit card risk type
complete_data['credit_cards_risk_type'] = complete_data
.apply(
    categorize_risk, axis=1)
# Select the relevant columns
credit_cards_risk = complete_data[['CustomerID', 'Educ
ationLevel', 'MaritalStatus',
                                   'AccountType', 'LoanAmount', 'Inte
restRate', 'CreditLimit', 'credit_cards_risk_type']]
```

Following is the output of *Project I*, Part 3**:**

| | CustomerID | EducationLevel | MaritalStatus | AccountType | LoanAmount | InterestRate | CreditLimit | credit_cards_risk_type |
|---|---|---|---|---|---|---|---|---|
| 0 | 34da4208-3b9e-486b-bd63-34381ad846dc | PhD | Widowed | Credit Card | 39117 | 7.63 | 6880 | High |
| 1 | 08d50394-bcf1-4537-861d-2eb481c938ee | Master | Divorced | Credit Card | 22027 | 5.06 | 4175 | High |
| 2 | a187a327-f9db-4fda-ac9c-05559737cdb3 | Master | Divorced | Loan | 46136 | 5.23 | 6347 | High |
| 3 | 1872823f-a112-45b3-84c5-96500b9ee8a7 | High School | Widowed | Savings | 8390 | 4.53 | 1285 | Low |
| 4 | c1c5156a-ffec-48da-97dc-64899a696a99 | High School | Married | Credit Card | 6735 | 1.74 | 7614 | High |

***Figure 11.4:*** *Data frame with customer bank details and credit card risk*

*Figure 11.4* is a data frame with a new column credit cards risk type, which indicates the risk level of the customer for issuing credit cards.

## Part 4: Predictive modeling

Now that we have a new data frame as shown in *Figure*

*11.4*, in it we will use regression to predict credit limits and classification to identify high-risk accounts with credit cards risk type as the output or target variable and others as inputs or predictors but before we apply logistic regression, we will encode categorical columns into integer numbers. The code snippet for encoding categorical values into numbers and applying predictive modelling as following:

```
# Load data : EducationLevel MaritalStatus AccountTyp
e Amount LoanType InterestRate CreditLimit
credit_cards = pd.read_csv('credit_cards_risk.csv')
# Mapping catagorrical variables into numerical values
as follow
education_levels = {"High School": 0, "Bachelor": 1, "Ma
ster": 2, "PhD": 3}
marital_status = {"Single": 0, "Married": 1, "Divorced": 2
, "Widowed": 3}
account_types = {"Checking": 0, "Savings": 1, "Credit Ca
rd": 2, "Loan": 3}
risk_types = {"Low": 0, "Medium": 1, "High": 2}
# Apply the mapping to the respective columns
credit_cards['EducationLevel'] = credit_cards['Education
Level'].map(
    education_levels)
credit_cards['MaritalStatus'] = credit_cards['MaritalStat
us'].map(
    marital_status)
credit_cards['AccountType'] = credit_cards['AccountTyp
e'].map(account_types)
credit_cards['credit_cards_risk_type'] = credit_cards['cre
dit_cards_risk_type'].map(
    risk_types)
# Prepare data for logistic regression
X = credit_cards[['EducationLevel', 'MaritalStatus', 'Acc
```

```
ountType',
                'LoanAmount', 'InterestRate', 'CreditLimit']]  # Predictor
y = credit_cards['credit_cards_risk_type']  # Response variable
# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
# Create a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)
# Predictions and evaluation
predictions = model.predict(X_test)
print(classification_report(y_test, predictions))
```

Following is the output of *Project I*, part 4**:**

The trained model evaluation matrices scores are as follows:

```
              precision    recall  f1-score   support

           0       0.85      0.34      0.48       116
           1       0.52      0.57      0.54       133
           2       0.73      0.90      0.81       251

    accuracy                           0.68       500
   macro avg       0.70      0.60      0.61       500
weighted avg       0.70      0.68      0.66       500
```

## Part 5: Use the predictive model above Part 4. Feed it user input and see predictions

Finally, we will take **EducationLevel**, **MaritalStatus**, **AccountType**, **LoanType**, **InterestRate**, **CreditLimit** as user input to see what **credit_card_risk_type** (high, low,

medium) the trained prediction model predicts. The code snippet to do so is as following:

```
# Define mappings for categorical input to integer encoding
education_levels = {"High School": 0, "Bachelor": 1, "Master": 2, "PhD": 3}
marital_status = {"Single": 0, "Married": 1, "Divorced": 2, "Widowed": 3}
account_types = {"Checking": 0, "Savings": 1, "Credit Card": 2, "Loan": 3}
risk_type_options = {0: 'Low', 1: 'Medium', 2: 'High'}
# Function to get user input and convert to encoded value
def get_user_input(prompt, category_dict):
    while True:
        response = input(prompt)
        if response in category_dict:
            return category_dict[response]
        else:
            print("Invalid entry. Please choose one of:",
                  list(category_dict.keys()))
# Function to get numerical input and validate it
def get_numerical_input(prompt):
    while True:
        try:
            value = float(input(prompt))
            return value
        except ValueError:
            print("Invalid entry. Please enter a valid number.")
# Collect inputs
education_level = get_user_input(
```

```
    "Enter Education Level (High School, Bachelor, Master, PhD): ", education_levels)
marital_status = get_user_input(
    "Enter Marital Status (Single, Married, Divorced, Widowed): ", marital_status)
account_type = get_user_input(
    "Enter Account Type (Checking, Savings, Credit Card, Loan): ", account_types)
loan_amount = get_numerical_input("Enter Loan Amount: ")
interest_rate = get_numerical_input("Enter Interest Rate: ")
credit_limit = get_numerical_input("Enter Credit Limit: ")
# Prepare the input data for prediction
input_data = pd.DataFrame({
    'EducationLevel': [education_level],
    'MaritalStatus': [marital_status],
    'AccountType': [account_type],
    'LoanAmount': [loan_amount],
    'InterestRate': [interest_rate],
    'CreditLimit': [credit_limit]
})
# Predict the risk type
prediction = model.predict(input_data)
print("Predicted Risk Type:", risk_type_options[prediction[0]])
```

Upon running the above *Project I*, part 5 snippet, you will be asked to provide the necessary input, based on which the predictive model will tell you if the risk is high, medium or low.

## Project II: Implementing data science and statistical analysis on health data

Project II explores the extensive capabilities of Python for implementing statistics concepts in the realm of health data analysis. Here we use a synthetic health data set generated for this tutorial. The dataset includes a variety of medical measurements reflecting common data types collected in health informatics. This synthetic dataset simulates realistic health records with 2500 entries containing metrics such **Body Mass Index (BMI),** glucose level, blood pressure, heart rate, cholesterol, hemoglobin, white blood cell count, and platelets. Each record also contains a unique patient ID and a binary health outcome indicating the presence or absence of a particular condition. This structure supports analyses ranging from basic descriptive statistics to complex machine learning models. The primary goal of this project is to provide hands-on experience and demonstrate how Python can be used to manipulate, analyze, and predict health outcomes based on statistical data. In Project II, we first perform exploratory data analysis to view and better understand the data set. Then we apply statistical analysis to look at the correlation and covariance between features. This is followed by inferential statistics where we compute t-statistics, p-value, and confidence interval for selected features if of interest. Finally, a statistical logistic regression model is trained to classify health outcomes as binary values representing good and bad health, and the results are evaluated.

## Part 1: Exploratory data analysis

In part 1 of Project II, we will examine data to understand distributions and relationships. We will visualize the distribution using histograms, box plots, and, then plot relationship between glucose level and cholesterol in a scatter plots. And then view summary of data through

measures of central tendency and variability. The code snippet is as follows:

```
# Load the data
data = pd.read_csv(
    '/workspaces/ImplementingStatisticsWithPython/notebooks/chapter11project/health/data/synthetic_health_data.csv')
# Define features for plots
features = ['Health_Outcome', 'Body_Mass_Index', 'Glucose_Level', 'Blood_Pressure',
            'Heart_Rate', 'Cholesterol', 'Haemoglobin', 'White_Blood_Cell_Count', 'Platelets']
# Plot histograms
fig, axs = plt.subplots(3, 3, figsize=(15, 10))
for ax, feature in zip(axs.flatten(), features):
    ax.hist(data[feature], bins=20, color='skyblue', edgecolor='black')
    ax.set_title(f'Histogram of {feature}')
plt.tight_layout()
plt.savefig('health_histograms.png', dpi=300, bbox_inches='tight')
plt.show()
```

Following is the output of *Project II*, part 1:

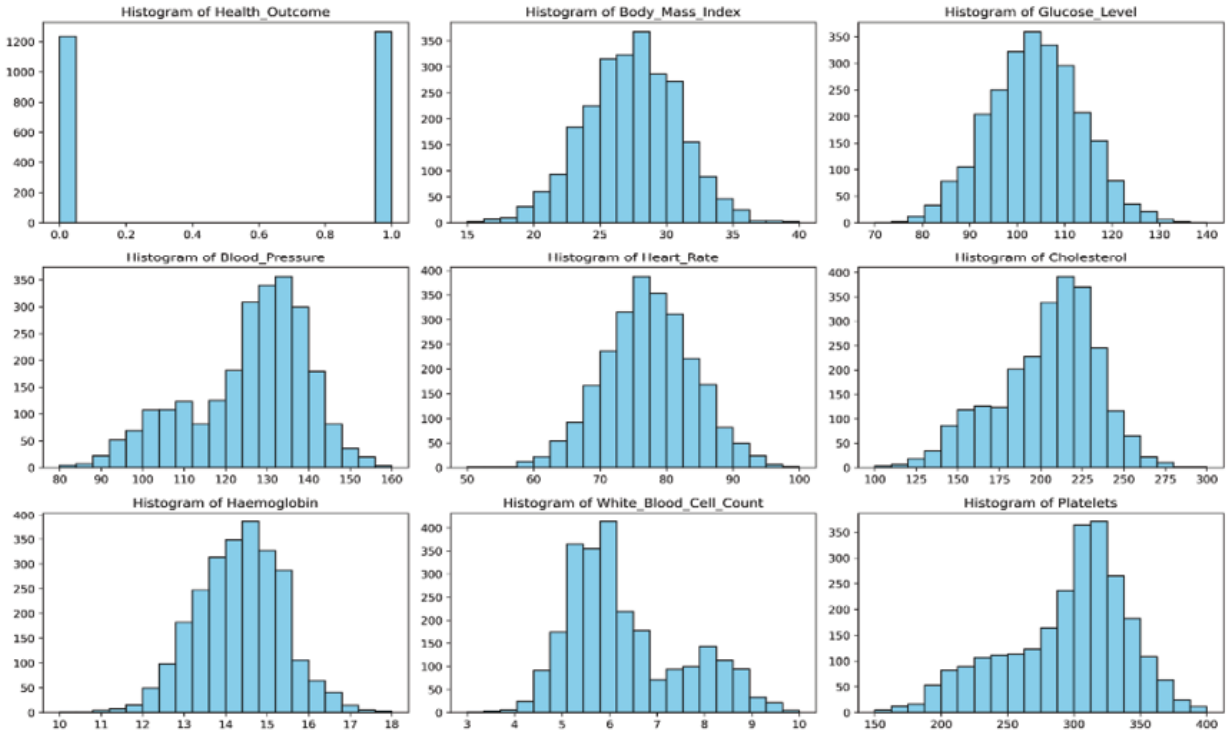


***Figure 11.5:*** *Distribution of customers across each feature*

*To see the distribution in box plot, code snippet is as follows:*

```
# Plot box plots
plt.figure(figsize=(12, 8))
sns.boxplot(data=data[features])
plt.xticks(rotation=45)
plt.title('Box Plot of Selected Variables')
plt.savefig('health_boxplot.png', dpi=300, bbox_inches='tight')
plt.show()
```

*Figure 11.6* shows the median, quartiles, and outliers, which represent the spread, skewness, and central tendency of the data in the box plot.

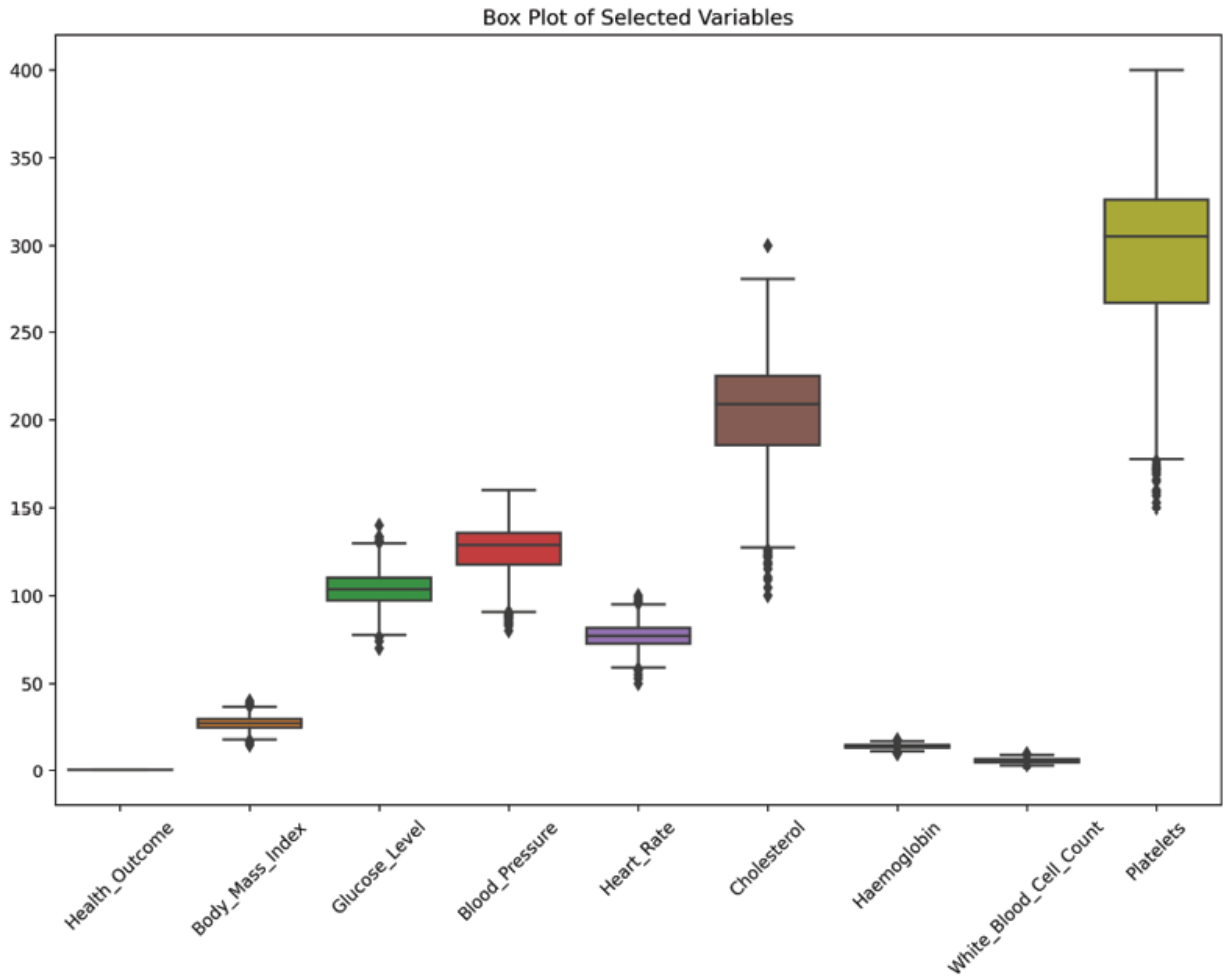


***Figure 11.6:*** *Box plot showing spread, skewness, and central tendency across each feature*

Then, to see the distribution in scatter plot, code snippet is as follows:

```
# Scatter plot of two variables
sns.scatterplot(x='Glucose_Level', y='Cholesterol', data=data)
plt.title('Scatter Plot of Glucose Level vs Cholesterol')
plt.savefig('health_scatterplot.png', dpi=300, bbox_inches='tight')
plt.show()
```

*Figure 11.7* shows that the majority of patients have glucose levels from 80 to 120 (milligrams per deciliter) and

cholesterol from 125 to 250 (milligrams per deciliter):

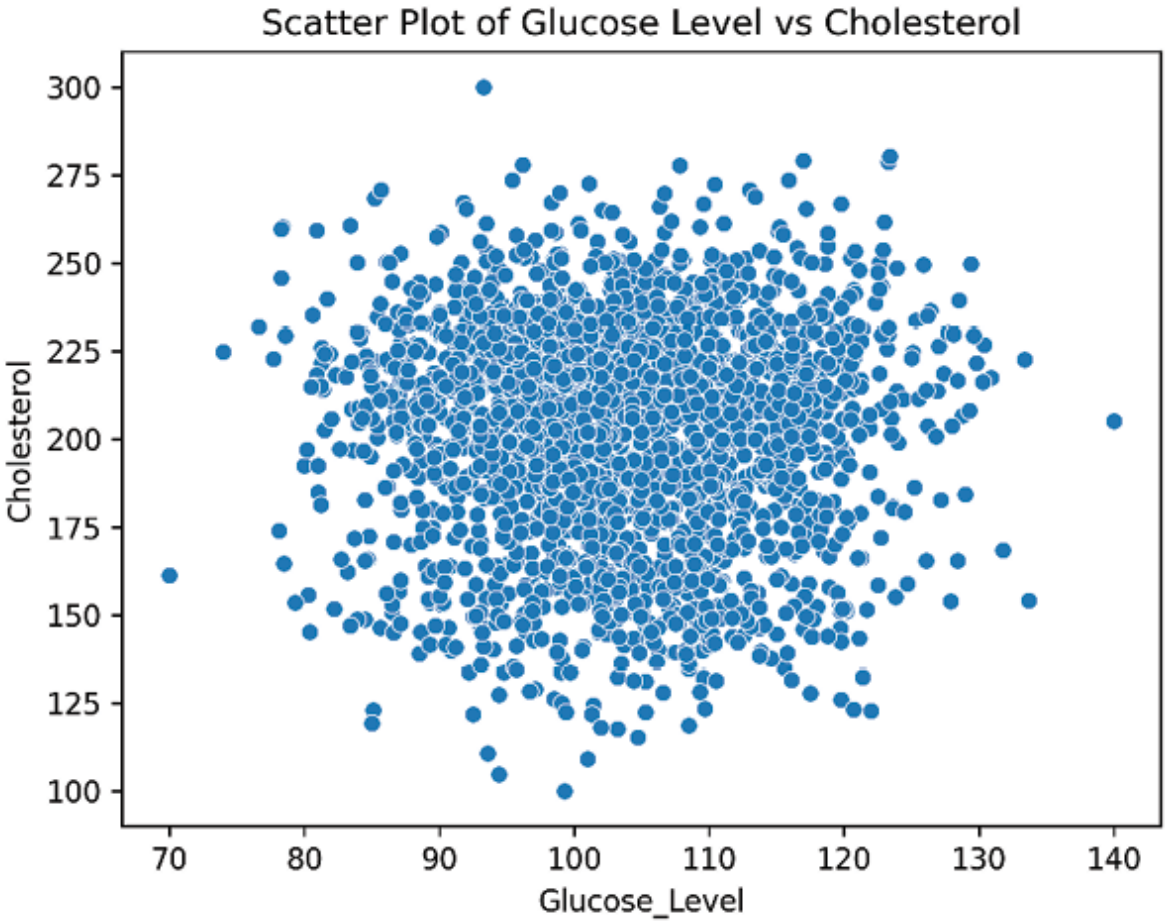


***Figure 11.7:*** *Scatter plot to view relationship between cholesterol and glucose level*

This following code displays the summary statistics of the features in data:

```
# Print descriptive statistics for the selected features
display(data[features].describe())
```

*Figure 11.8* shows platelets variable has wide range of values, with a minimum of 150 and a maximum of 400. This suggests considerable variation in platelet counts within the dataset, which may be important for understanding potential health outcomes.

| | Health_Outcome | Body_Mass_Index | Glucose_Level | Blood_Pressure | Heart_Rate | Cholesterol | Haemoglobin | White_Blood_Cell_Count | Platelets |
|---|---|---|---|---|---|---|---|---|---|
| count | 2500.000000 | 2500.000000 | 2500.000000 | 2500.000000 | 2500.000000 | 2500.000000 | 2500.000000 | 2500.000000 | 2500.000000 |
| mean | 0.506000 | 27.386360 | 103.778120 | 125.760120 | 77.287960 | 204.128280 | 14.297880 | 6.290240 | 294.893600 |
| std | 0.500064 | 3.508407 | 9.687728 | 14.084897 | 6.773203 | 30.023524 | 1.047151 | 1.226606 | 45.792266 |
| min | 0.000000 | 15.000000 | 70.000000 | 80.000000 | 50.000000 | 100.000000 | 10.000000 | 3.000000 | 150.000000 |
| 25% | 0.000000 | 25.000000 | 97.175000 | 117.775000 | 72.700000 | 185.900000 | 13.600000 | 5.400000 | 266.675000 |
| 50% | 1.000000 | 27.500000 | 103.750000 | 128.800000 | 77.300000 | 209.100000 | 14.300000 | 6.000000 | 305.300000 |
| 75% | 1.000000 | 29.800000 | 110.400000 | 135.800000 | 81.800000 | 225.025000 | 15.000000 | 7.000000 | 326.150000 |
| max | 1.000000 | 40.000000 | 140.000000 | 160.000000 | 100.000000 | 300.000000 | 18.000000 | 10.000000 | 400.000000 |

***Figure 11.8:*** *Summary statistics of selected features*

## Part 2: Statistical analysis

Here we will view relationships between variables using covariance and correlation and look out for outliers using z score measure. The code is as follows:

```
# Select features for analysis
features = ['Health_Outcome', 'Body_Mass_Index', 'Glucose_Level', 'Blood_Pressure',
            'Heart_Rate', 'Cholesterol', 'Haemoglobin', 'White_Blood_Cell_Count', 'Platelets']
# Analyzing relationships between variables using covariance and correlation.
# Correlation matrix
correlation_matrix = data[features].corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True)
plt.title('Correlation Matrix')
plt.show()
```

The above code illustrates correlations between chosen features. A correlation coefficient of +1 denotes high positive correlation, indicating that as one feature increases, the other also increases, and vice versa. Conversely, a coefficient of -1 signifies high negative correlation, suggesting that as one feature increases, the other decreases, and vice versa as following:

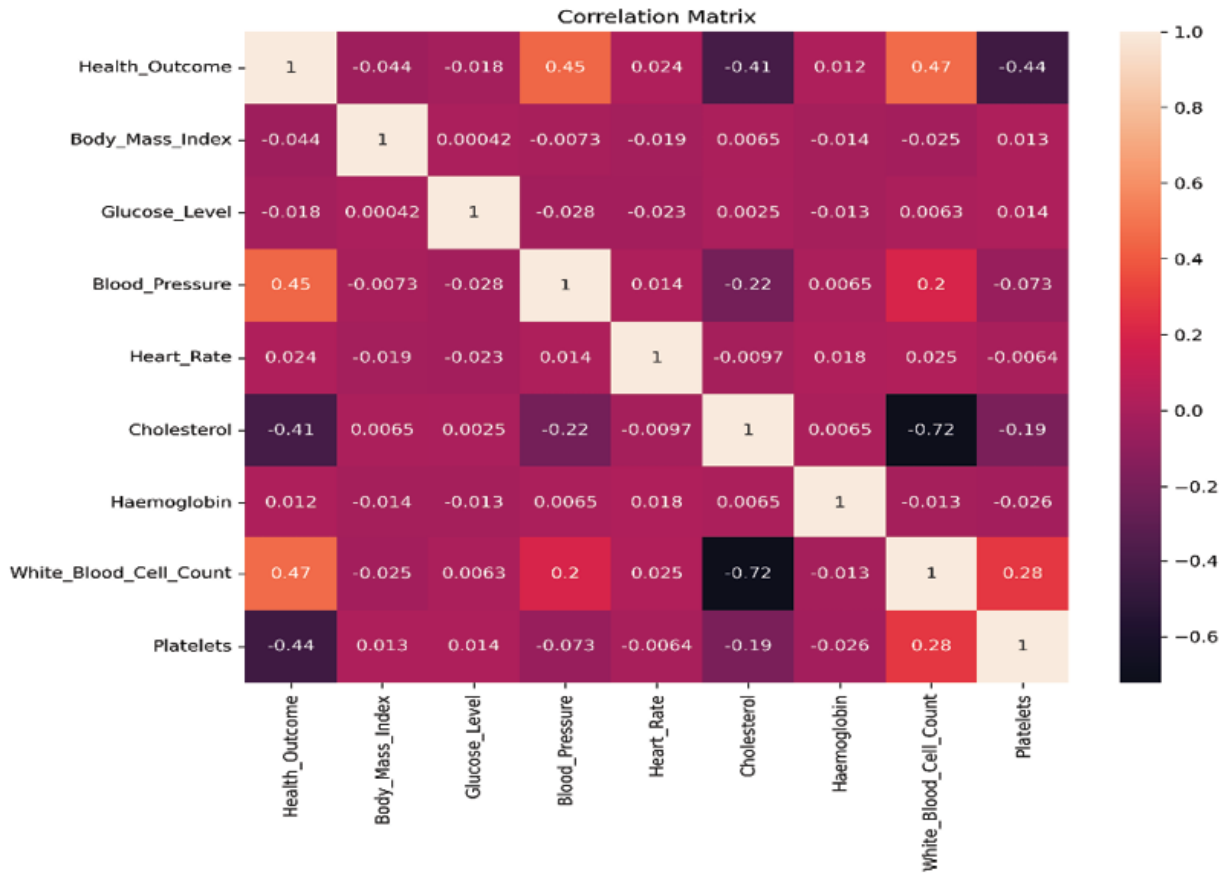


***Figure 11.9:*** *Correlation matrix of features, color intensity represents level of correlation*

Again, we employ a covariance matrix to observe covariance values. A high positive covariance indicates that both variables move in the same direction as one increases, the other tends to increase and vice versa. Conversely, a high negative covariance implies that both variables move in opposite directions as one increases, the other tends to decrease, and vice versa. The following code illustrates the covariance between features:

```
# Covariance matrix
covariance_matrix = data[features].cov()
print("Covariance Matrix:")
display(covariance_matrix)
```

Then, using the following code, we will calculate the z-scores for each element in the dataset, z-score quantifies

how many standard deviations a data point is from the dataset's mean. Here we use the condition *abs_z_scores > 1*. This metric is crucial for identifying outliers, as it provides a standardized way to detect outliers. As the output, it does not detect any outliers:

```
# Identifying outliers and understanding their impact.
# Z-score for outlier detection
z_scores = zscore(data)
abs_z_scores = np.abs(z_scores)
outliers = (abs_z_scores > 1).all(axis=1)
data_outliers = data[outliers]
print("Detected Outliers:")
print(data_outliers)
```

## Part 3: Inferential statistics

Now, we will use statistical methods to infer population characteristics from glucose level data categorized by health outcomes. We will begin by performing a t-test to compare the mean glucose levels between groups with different health outcomes, yielding a t-statistic and p-value to assess the significance of differences. Following this, we will calculate the 95% confidence interval for the overall mean glucose level, providing a range that likely includes the true mean with 95% certainty. These steps help determine the relationship between health outcomes and glucose levels and estimate the mean glucose level's variability as follows:

```
# Apply T-test
group1 = data[data['Health_Outcome'] == 0]['Glucose_Level']
group2 = data[data['Health_Outcome'] == 1]['Glucose_Level']
t_stat, p_val = ttest_ind(group1, group2)
```

```
print(f"T-statistic: {t_stat}, P-value: {p_val}")
# Confidence interval for the mean of a column
ci_low, ci_upp = norm.interval(
    alpha=0.95, loc=data['Glucose_Level'].mean(), scale=
data['Glucose_Level'].std())
print(
    f"95% confidence interval for the mean glucose level:
({ci_low}, {ci_upp})")
```

Following is the output of *Project II,* part 3:

A T-statistic of 0.92 indicates a moderate difference between the mean glucose levels of two groups. The P-value of 0.36 indicates that there is a 36% chance of observing such a difference if there were no true difference between the groups and the confidence interval score suggests that we are 95% confident that the true mean glucose level for Group 1 is between 84.79 and 122.77, and similarly for Group 2 as follows:

```
T-statistic: 0.9204677863057696, P-
value: 0.3574172393450691
95% confidence interval for the mean glucose level:
(84.79052199831503, 122.76571800168497)
```

## Part 4: Statistical machine learning

Finally, we will train a logistic regression model using input features ( **Body_Mass_Index**, **Glucose_Level**, **Blood_Pressure**, **Heart_Rate**, **Cholesterol**, **Haemoglobin**, **White_Blood_Cell_Count**, **Platelets**) to predict binary class outcomes (**Health_Outcome**) and evaluates the model's accuracy, displays a confusion matrix for insight into performance, and plots a **Receiver-Operating Characteristic Curve (ROC)** curve to assess its ability to classify instances as follows:

```
X = data.drop(['Health_Outcome', 'Patient_ID'], axis=1)
```

```
y = data['Health_Outcome']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)
model = LogisticRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
# Accuracy and confusion matrix
print("Accuracy:", model.score(X_test, y_test))
print("Confusion Matrix:")
print(confusion_matrix(y_test, predictions))
# ROC Curve and AUC
probs = model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, probs)
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2,
         label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve'
)
plt.legend(loc="lower right")
plt.show()
```

Following is the output of *Project II*, part 4:

As a result, we obtained a trained model with an accuracy of 94.26%, which means that the model correctly predicts the outcome about 94.26% of the time and the receiver operating characteristics curve value of 97% indicates that the model has a high true positive rate and a low false positive rate, which means strong predictive ability as follows:

```
Accuracy: 0.9426666666666667
Confusion Matrix:
[[331  28]
 [ 15 376]]
```

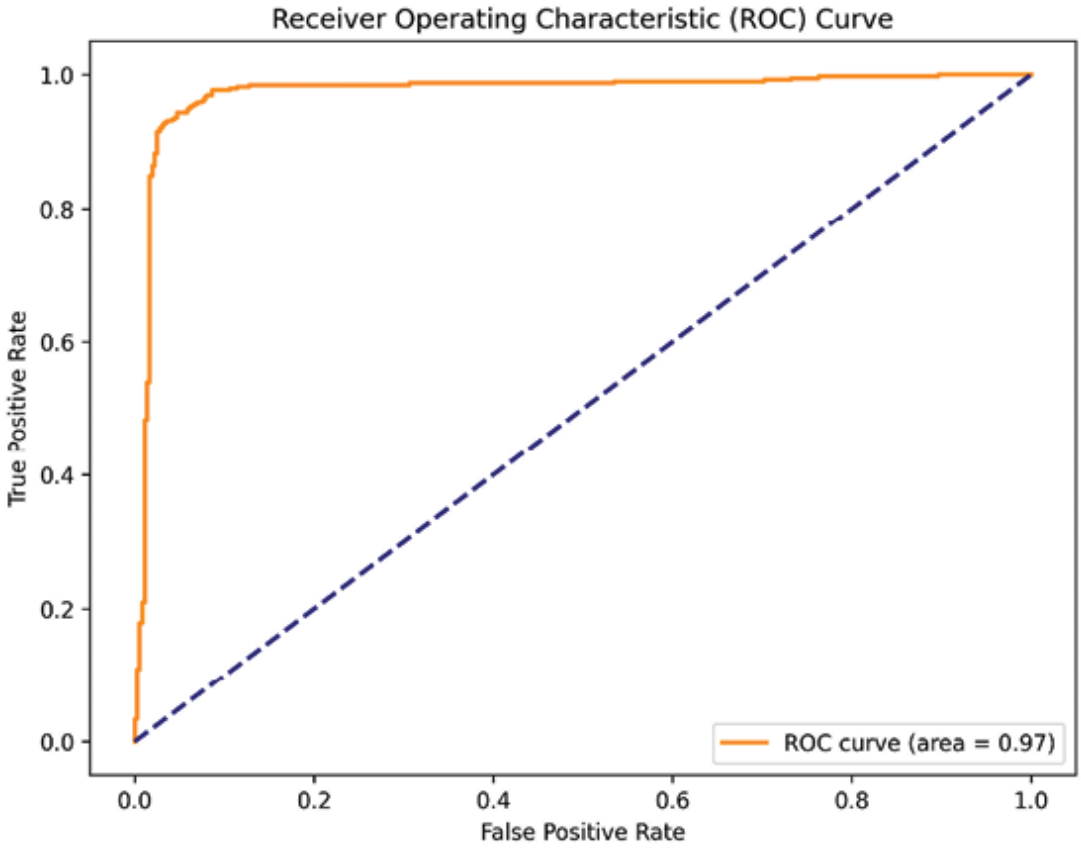


***Figure 11.10:*** *Receiver operating characteristic curve of the health outcome prediction model*

## Conclusion

This chapter provided a hands-on experience in the practical application of data science and statistical analysis in two critical sectors: banking and healthcare. Using synthetic data, the chapter demonstrated how the theories, methods, and techniques covered throughout the book can be skillfully applied to real-world contexts. However, the use of statistics, data science, and Python programming extends

far beyond these examples. In banking, additional applications include fraud detection and risk assessment, customer segmentation, and forecasting. In healthcare, applications extend to predictive modelling for patient outcomes, disease surveillance and public health management, and improving operational efficiency in healthcare systems.

Despite these advances, the real-world use of data requires careful consideration of ethical, privacy, and security issues, which are paramount and must always be carefully addressed. In addition, the success of statistical applications is highly dependent on the quality and granularity of the data, making data quality and management equally critical. With ongoing technological advancements and regulatory changes, there is a constant need to learn and adapt new methodologies and tools. This dynamic nature of data science requires practitioners to remain current and flexible to effectively navigate the evolving landscape.

## Join our book’s Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

**https://discord.bpbonline.com**